Solaris

libkstat

<kstat.h>

<sys/kstat.h>

http://docs.sun.com/app/docs/doc/806-0631/6j9vl9p85?a=view

e.g. from the link above
Using libkstat

The kstat library, libkstat, defines the user interface (API) to the system's kstat facility.

You begin by opening libkstat with kstat_open(3KSTAT), which returns a pointer to a fully initialized kstat control structure. This is your ticket to subsequent libkstat operations:

```
typedef struct kstat_ctl {
kid_t kc_chain_id; /* current kstat chain ID */
kstat_t *kc_chain; /* pointer to kstat chain */
int kc_kd; /* /dev/kstat descriptor */
} kstat_ctl_t;
```

Only the first two fields, kc_chain_id and kc_chain, are of interest to libkstat clients. (kc_kd is the descriptor for /dev/kstat, the kernel statistics driver. libkstat functions are built on top of /dev/kstat ioctl(2) primitives. Direct interaction with /dev/kstat is strongly discouraged, since it is not a public interface.)

kc_chain points to your copy of the kstat chain. You typically walk the chain to find and process a certain kind of kstat. For example, to display all I/O kstats:

```
kstat_ctl_t *kc;
kstat_t *ksp;
kstat_io_t kio;

kc = kstat_open();
for (ksp = kc->kc_chain; ksp != NULL; ksp = ksp->ks_next) {
if (ksp->ks_type == KSTAT_TYPE_IO) {
kstat_read(kc, ksp, &kio);
my_io_display(kio);
}
}
```

kc_chain_id is the kstat chain ID, or KCID, of your copy of the kstat chain. See kstat_chain_update(3KSTAT) for an explanation of KCIDs.

/proc/<pid>

/proc/<pid>/psinfo

Needs to be read as root, else permission denied on other user processes.

<sys/procfs.h>

Read from file into corresponding structure

e.g., (pseudocode without error checking)

```
psinfo_t my_psinfo;
int fd = open ("/proc/<pid>/psinfo". O_RDONLY);
ssize_t bytes_read = read(fd, &my_psinfo, sizeof(psinfo_t));
Now use my_psinfo.pr_pid etc.
```

/proc/<pid>/status

typedef struct pstatus {

1. int pr_flags; /* flags (see below) */
2. int pr_nlwp; /* number of lwps in the process */
3. pid_t pr_pid; /* process id */
4. pid_t pr_ppid; /* parent process id */
5. pid_t pr_pgid; /* process group id */
6. pid_t pr_sid; /* session id */
7. id_t pr_aslwpid; /* lwp id of the aslwp, if any */
8. id_t pr_agentid; /* lwp id of the /proc agent lwp, if any */
9. sigset_t pr_sigpend; /* set of process pending signals */
10. uintptr_t pr_brkbase; /* address of the process heap */
11. size_t pr_brksize; /* size of the process heap, in bytes */
12. uintptr_t pr_stkbase; /* address of the process stack */
13. size_t pr_stksize; /* size of the process stack, in bytes */
14. timestruc_t pr_utime; /* process user cpu time */
15. timestruc_t pr_stime; /* process system cpu time */
16. timestruc_t pr_cutime; /* sum of children's user times */
17. timestruc_t pr_cstime; /* sum of children's system times */
18. sigset_t pr_sigtrace; /* set of traced signals */
19. fltset_t pr_flttrace; /* set of traced faults */
20. sysset_t pr_sysentry; /* set of system calls traced on entry */
21. sysset_t pr_sysexit; /* set of system calls traced on exit */
22. char pr_dmodel; /* data model of the process (see below) */
23. char pr_pad[3];
24. taskid_t pr_taskid; /* task id */
25. projid_t pr_projid; /* project id */
26. int pr_filler[17]; /* reserved for future use */
27. lwpstatus_t pr_lwp; /* status of the representative lwp */

} pstatus_t;

See <sys/procfs.h> for #defines for pr_flags

typedef struct psinfo {

1. int pr_flag; /* process flags */
2. int pr_nlwp; /* number of lwps in process */
3. pid_t pr_pid; /* unique process id */
4. pid_t pr_ppid; /* process id of parent */
5. pid_t pr_pgid; /* pid of process group leader */
6. pid_t pr_sid; /* session id */
7. uid_t pr_uid; /* real user id */
8. uid_t pr_euid; /* effective user id */
9. gid_t pr_gid; /* real group id */
10. gid_t pr_egid; /* effective group id */
11. uintptr_t pr_addr; /* address of process */
12. size_t pr_size; /* size of process image in Kbytes */
13. size_t pr_rssize; /* resident set size in Kbytes */
14. size_t pr_pad1;
15. dev_t pr_ttydev; /* controlling tty device (or PRNODEV) */
16. /* The following percent numbers are 16-bit binary */
17. /* fractions [0 .. 1] with the binary point to the */
18. /* right of the high-order bit (1.0 == 0x8000) */
19. ushort_t pr_pctcpu; /* % of recent cpu time used by all lwps */
20. ushort_t pr_pctmem; /* % of system memory used by process */
21. timestruc_t pr_start; /* process start time, from the epoch */
22. timestruc_t pr_time; /* usr+sys cpu time for this process */
23. timestruc_t pr_ctime; /* usr+sys cpu time for reaped children */
24. char pr_fname[PRFNSZ]; /* name of execed file */
25. char pr_psargs[PRARGSZ]; /* initial characters of arg list */
26. int pr_wstat; /* if zombie, the wait() status */
27. int pr_argc; /* initial argument count */
28. uintptr_t pr_argv; /* address of initial argument vector */
29. uintptr_t pr_envp; /* address of initial environment vector */
30. char pr_dmodel; /* data model of the process */
31. char pr_pad2[3];
32. taskid_t pr_taskid; /* task id */
33. projid_t pr_projid; /* project id */
34. int pr_filler[5]; /* reserved for future use */
35. lwpsinfo_t pr_lwp; /* information for representative lwp */

} psinfo_t;

/proc/<pid>/lwp

/proc/<pid>/lwp/<lwpid>/lwpstatus

typedef struct lwpstatus {

1. int pr_flags; /* flags (see below) */
2. id_t pr_lwpid; /* specific lwp identifier */
3. short pr_why; /* reason for lwp stop, if stopped */
4. short pr_what; /* more detailed reason */
5. short pr_cursig; /* current signal, if any */
6. short pr_pad1;
7. siginfo_t pr_info; /* info associated with signal or fault */
8. sigset_t pr_lwppend; /* set of signals pending to the lwp */
9. sigset_t pr_lwphold; /* set of signals blocked by the lwp */
10. struct sigaction pr_action; /* signal action for current signal */
11. stack_t pr_altstack; /* alternate signal stack info */
12. uintptr_t pr_oldcontext; /* address of previous ucontext */
13. short pr_syscall; /* system call number (if in syscall) */
14. short pr_nsysarg; /* number of arguments to this syscall */
15. int pr_errno; /* errno for failed syscall, 0 if successful */
16. long pr_sysarg[PRSYSARGS]; /* arguments to this syscall */
17. long pr_rval1; /* primary syscall return value */
18. long pr_rval2; /* second syscall return value, if any */
19. char pr_clname[PRCLSZ]; /* scheduling class name */
20. timestruc_t pr_tstamp; /* real-time time stamp of stop */
21. timestruc_t pr_utime; /* lwp user cpu time */
22. timestruc_t pr_stime; /* lwp system cpu time */
23. int pr_filler[12 - 2 * sizeof (timestruc_t) / sizeof (int)];
24. uintptr_t pr_ustack; /* address of stack boundary data (stack_t) */
25. ulong_t pr_instr; /* current instruction */
26. prgregset_t pr_reg; /* general registers */
27. prfpregset_t pr_fpreg; /* floating-point registers */

} lwpstatus_t;

/proc/<pid>/lwp/<lwpid>/lwpsinfo

typedef struct lwpsinfo {

1. int pr_flag; /* lwp flags */
2. id_t pr_lwpid; /* lwp id */
3. uintptr_t pr_addr; /* internal address of lwp */
4. uintptr_t pr_wchan; /* wait addr for sleeping lwp */
5. char pr_stype; /* synchronization event type */
6. char pr_state; /* numeric lwp state */
7. char pr_sname; /* printable character for pr_state */
8. char pr_nice; /* nice for cpu usage */
9. short pr_syscall; /* system call number (if in syscall) */
10. char pr_oldpri; /* pre-SVR4, low value is high priority */
11. char pr_cpu; /* pre-SVR4, cpu usage for scheduling */
12. int pr_pri; /* priority, high value is high priority */
13. /* The following percent number is a 16-bit binary */
14. /* fraction [0 .. 1] with the binary point to the */
15. /* right of the high-order bit (1.0 == 0x8000) */
16. ushort_t pr_pctcpu; /* % of recent cpu time used by this lwp */
17. ushort_t pr_pad;
18. timestruc_t pr_start; /* lwp start time, from the epoch */
19. timestruc_t pr_time; /* usr+sys cpu time for this lwp */
20. char pr_clname[PRCLSZ]; /* scheduling class name */
21. char pr_name[PRFNSZ]; /* name of system lwp */
22. processorid_t pr_onpro; /* processor which last ran this lwp */
23. processorid_t pr_bindpro; /* processor to which lwp is bound */
24. psetid_t pr_bindpset; /* processor set to which lwp is bound */
25. int pr_filler[5]; /* reserved for future use */

} lwpsinfo_t;